Task 1 Report

Summaries of your attempt to setup your environment, including details of your requirements file

To setup the virtual machine, the required packages had to be installed to ensure that the stock prediction projects were going to effectively. This required installing the packages using 'pip' in the PyCharm virtual machine terminal

```
(venv) (base) conzo@Connors-MBP-2 Week 2 Portfolio  % pip install pandas
```

Figure 1: Installing pandas in the virtual environment

The following packages were installed and outlined in the requirements file:

- pandas
- numpy
- tensorflow
- matplotlib
- pandas
- yahoo_fin
- scikit-learn → sklearn is deprecated and no longer used
- yfinance

The requirements file contains the correct syntax and names for these packages to ensure that when another user intends to download the stock prediction project, they install the correct packages.

Summaries of your attempts to test the provided code bases (v0.1 and P1) with screenshots.

Testing code base v0.1:

```
Epoch 23/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 0.0054
Epoch 24/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0049
Epoch 25/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 0.0053
[********************100%%**********************]  1 of 1 completed
```

Figure 2: Final Epochs for training

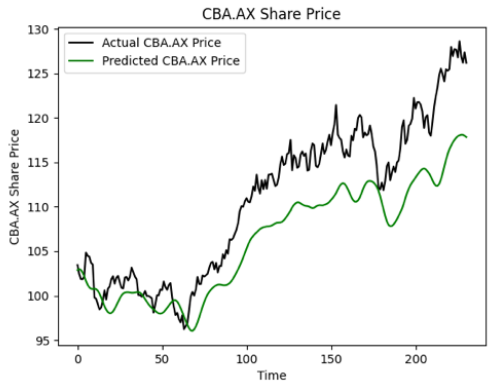

Figure 3: Share Price Data Graph

The line...

```
from keras import Sequential
```

Had to be altered to ensure that the code ran without errors. Since the version of Tensorflow is updated beyond version 2.16, the usage of Keras works as a standalone package, so tensorflow.keras was replaced with keras.
Code functioned as expected, however was given the warning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
super().__init__(**kwargs)

The code ran efficiency, however due to the lack of days for the prediction_days, I believe the data wasn't very accurate, although it did manage to copy a similar path in the share price.

Testing code base P1:

```
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step - loss: 4.1362e-04 - mean_absolute_error: 0.0163
Epoch 495: val_loss did not improve from 0.00039
85/85 ━━━━━━━━━━━━━━━━━━━━ 17s 204ms/step - loss: 4.1349e-04 - mean_absolute_error: 0.0163 - val_loss: 4.3036e-04 - val_mean_absolute_error: 0.0139
Epoch 496/500
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step - loss: 4.3206e-04 - mean_absolute_error: 0.0165
Epoch 496: val_loss did not improve from 0.00039
85/85 ━━━━━━━━━━━━━━━━━━━━ 16s 190ms/step - loss: 4.3165e-04 - mean_absolute_error: 0.0165 - val_loss: 4.2974e-04 - val_mean_absolute_error: 0.0141
Epoch 497/500
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step - loss: 4.4252e-04 - mean_absolute_error: 0.0170
Epoch 497: val_loss did not improve from 0.00039
85/85 ━━━━━━━━━━━━━━━━━━━━ 18s 208ms/step - loss: 4.4219e-04 - mean_absolute_error: 0.0170 - val_loss: 4.7692e-04 - val_mean_absolute_error: 0.0151
Epoch 498/500
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step - loss: 4.1941e-04 - mean_absolute_error: 0.0167
Epoch 498: val_loss did not improve from 0.00039
85/85 ━━━━━━━━━━━━━━━━━━━━ 17s 203ms/step - loss: 4.1920e-04 - mean_absolute_error: 0.0167 - val_loss: 4.0374e-04 - val_mean_absolute_error: 0.0139
Epoch 499/500
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step - loss: 4.2195e-04 - mean_absolute_error: 0.0167
Epoch 499: val_loss did not improve from 0.00039
85/85 ━━━━━━━━━━━━━━━━━━━━ 17s 195ms/step - loss: 4.2172e-04 - mean_absolute_error: 0.0167 - val_loss: 4.5962e-04 - val_mean_absolute_error: 0.0143
Epoch 500/500
85/85 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step - loss: 4.3473e-04 - mean_absolute_error: 0.0169
Epoch 500: val_loss did not improve from 0.00039
85/85 ━━━━━━━━━━━━━━━━━━━━ 17s 199ms/step - loss: 4.3441e-04 - mean_absolute_error: 0.0169 - val_loss: 4.0687e-04 - val_mean_absolute_error: 0.0138

Process finished with exit code 0
```

Figure 4: Final Epochs for the model's training

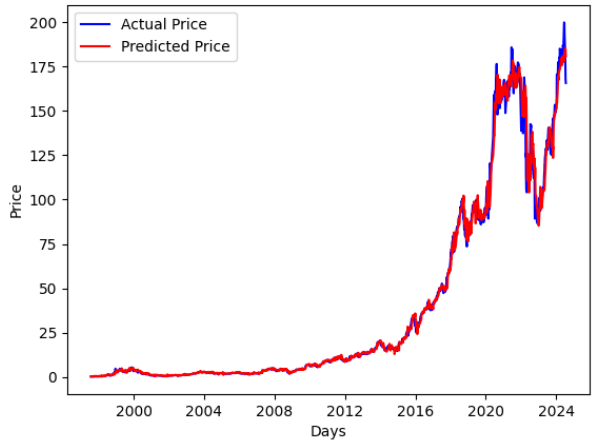

Figure 5: Amazon stock price prediction based off training

The key parameters of this model were:

- Epochs (EPOCHS): 500
- Batch Size (BATCH_SIZE): 64

This took a large amount of time, roughly between 1hrs – 2hrs for the model to train itself. The test_size parameter was 0.2, meaning that 20% of the data was used for testing and 80% was training.

This provided very accurate data when comparing the actual price to the predicted price.

Much of the code needed to be updated to allow for the packages to be configured properly. This required the keras packages to be reordered.

The create_model function also had to be modified to use input_shape instead of batch_input_shape, ensuring that it only specified the shape of the input data, excluding the batch size.

Summary of your understanding of the initial code base v0.1

- yfinance is utilised to download the stock price data for the company within a specified range.
- PRICE_VALUE = "Close" means that the model is using data to be used for training and prediction is the closing price of the stock.
- The training utilises the last 60 days to predict the next day's price.
- Prices are normalised to the range of 0 to 1 using the MinMaxScaler imported from scikit-learn
- reshape(-1, 1) converts the data from a 1D array of stock closing prices to a 2D array with one column for the MinMaxScaler to normalize the data.

```
PRICE_VALUE = "Close"

scaler = MinMaxScaler(feature_range=(0, 1))
# Note that, by default, feature_range=(0, 1). Thus, if you want a different
# feature_range (min,max) then you'll need to specify it here
scaled_data = scaler.fit_transform(data[PRICE_VALUE].values.reshape(-1, 1))
# Flatten and normalise the data
# First, we reshape a 1D array(n) to 2D array(n,1)
```

- Initializes the lists for the training data to store the input sequences (x_train) and the target values (y_train).
- Then the data is converted back to 1D for simplicity.
- Then the x_train and y_train lists are populated with past data based on their corresponding target values. Looped from 'PREDICITION DAYS' to the end of 'scaled data'.

```
# Number of days to look back to base the prediction
PREDICTION_DAYS = 60 # Original

# To store the training data
x_train = []
y_train = []

scaled_data = scaled_data[:,0] # Turn the 2D array back to a 1D array
# Prepare the data
for x in range(PREDICTION_DAYS, len(scaled_data)):
    x_train.append(scaled_data[x-PREDICTION_DAYS:x])
    y_train.append(scaled_data[x])
```

- Data is reshaped to 3D format required by the LSTM network.

```
# Convert them into an array
x_train, y_train = np.array(x_train), np.array(y_train)
# Now, x_train is a 2D array(p,q) where p = len(scaled_data) - PREDICTION_DAYS
# and q = PREDICTION_DAYS; while y_train is a 1D array(p)

x_train = np.reshape(x_train, newshape: (x_train.shape[0], x_train.shape[1], 1))
# We now reshape x_train into a 3D array(p, q, 1); Note that x_train
# is an array of p inputs with each input being a 2D array
```

- Initializes a Sequential model, which is a linear stack of layers.
- Three LTSM (Long Short-Term Memory) layers are added with dropout layers in-between which prevent overfitting. These layers remember sequences of data over time. The final layer is a Dense layer which connects each neuron in the previous layer to the current one, outputting the final predicted stock price after the LTSM layers have processed the sequential data.

```python
model = Sequential() # Basic neural network
# See: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
# for some useful examples

model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))

model.add(Dropout(0.2))
# The Dropout layer randomly sets input units to 0 with a frequency of
# rate (= 0.2 above) at each step during training time, which helps
# prevent overfitting (one of the major problems of ML).

model.add(LSTM(units=50, return_sequences=True))
# More on Stacked LSTM:
# https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))

model.add(Dense(units=1))
# Prediction of the next closing value of the stock price

# We compile the model by specify the parameters for the model
# See lecture Week 6 (COS30018)
model.compile(optimizer='adam', loss='mean_squared_error')
# The optimizer and loss are two important parameters when building an
# ANN model. Choosing a different optimizer/loss can affect the prediction
# quality significantly. You should try other settings to learn; e.g.

# optimizer='rmsprop'/'sgd'/'adadelta'/...
# loss='mean_absolute_error'/'huber_loss'/'cosine_similarity'/...

# Now we are going to train this model with our training data
# (x_train, y_train)
model.fit(x_train, y_train, epochs=25, batch_size=32)
```